NAME

panic - bring down system on fatal error

SYNOPSIS

#include <sys/types.h>
#include <sys/systm.h>

*extern char *panicstr;*

void
panic(*const char *fmt, ...*);

void
vpanic(*const char *fmt, va_list ap*);

KERNEL_PANICKED();

DESCRIPTION

The **panic**() and **vpanic**() functions terminate the running system. The message *fmt* is a printf(3) style format string. The message is printed to the console and *panicstr* is set pointing to the address of the message text. This can be retrieved from a core dump at a later time.

Upon entering the **panic**() function the panicking thread disables interrupts and calls critical_enter(9). This prevents the thread from being preempted or interrupted while the system is still in a running state. Next, it will instruct the other CPUs in the system to stop. This synchronizes with other threads to prevent concurrent panic conditions from interfering with one another. In the unlikely event of concurrent panics, only one panicking thread will proceed.

Control will be passed to the kernel debugger via **kdb_enter**(). This is conditional on a debugger being installed and enabled by the *debugger_on_panic* variable; see ddb(4) and gdb(4). The debugger may initiate a system reset, or it may eventually return.

Finally, kern_reboot(9) is called to restart the system, and a kernel dump will be requested. If **panic**() is called recursively (from the disk sync routines, for example), **kern_reboot**() will be instructed not to sync the disks.

The **vpanic**() function implements the main body of **panic**(). It is suitable to be called by functions which perform their own variable-length argument processing. In all other cases, **panic**() is preferred.

The **KERNEL_PANICKED**() macro is the preferred way to determine if the system has panicked. It

returns a boolean value. Most often this is used to avoid taking an action that cannot possibly succeed in a panic context.

EXECUTION CONTEXT

Once the panic has been initiated, code executing in a panic context is subject to the following restrictions:

- Single-threaded execution. The scheduler is disabled, and other CPUs are stopped/forced idle. Functions that manipulate the scheduler state must be avoided. This includes, but is not limited to, wakeup(9) and sleepqueue(9) functions.
- Interrupts are disabled. Device I/O (e.g. to the console) must be achieved with polling.
- Dynamic memory allocation cannot be relied on, and must be avoided.
- Lock acquisition/release will be ignored, meaning these operations will appear to succeed.
- Sleeping on a resource is not strictly prohibited, but will result in an immediate return from the sleep function. Time-based sleeps such as pause(9) may be performed as a busy-wait.

RETURN VALUES

The **panic**() and **vpanic**() functions do not return.

SEE ALSO

printf(3), ddb(4), gdb(4), KASSERT(9), kern_reboot(9)